MULTIMEDIA UNIVERSITY

STUDENT ID NO

MULTIMEDIA UNIVERSITY

FINAL EXAMINATION

TRIMESTER 1, 2019/2020

TDS2111 - DATA STRUCTURES AND ALGORITHMS

(All sections / Groups)

19 OCTOBER 2019
02:30 p.m. - 04:30 p.m.
(2 Hours)

INSTRUCTIONS TO STUDENTS

1. This question paper consists of 9 pages (excluding cover page) with 4 structured questions.

2. Attempt **ALL** questions. All questions carry equal marks and the distribution of the marks for each question is given.

3. Please write all your answers in the answer booklet provided.

Question 1

(a) Write the outputs of the following program with respect to the inputs x and y given below:

(i) `x = -999, y = 0;`
(ii) `x = 1, y = 999;`
(iii) `x = 5, y = 15;`

[3 marks]

```
#include <iostream>
using namespace std;

long refunc(int x, int y)
{
    if(x == 0 || y == 0)
        return 1;
    else if(x < 0 || y < 0)
        return 0;
    else
        return refunc(x - 1, y - 5) + x;
}

int main()
{
    int x, y;
    cin >> x >> y;
    cout << refunc(x, y);
    return 0;
}
```

Continued .....

(b) Provided with the following singly linked-list of seven (7) nodes:

```
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7
```

Complete the `myLinkedList()` function, such that it displays the data element belonging to the alternate nodes of the linked-list as follows:

```
1 -> 3 -> 5 -> 7
```

[2 marks]

```
...

/* A Linked-List Node */
struct Node
{
    int data;
    Node *next;
};

// navigatePtr is a navigation pointer being initialized
// to the head of a linked-list during run-time
void myLinkedList( Node* navigatePtr )
{
    if(navigatePtr == NULL)
        return;

    // TODO : Question 1(b)(i)
    // Print the data element of the current
    // linked-list node
    // ...

    if( navigatePtr ->next != NULL )
        // TODO : Question 1(b)(ii)
        // Include a recursive function call to
        // myLinkedList( );
}

...
```

Continued .....

(c) The following `evaluatePostfix()` function is an array-implemented stack for evaluating a postfix expression.

Complete the missing parts in your answer booklet. **[5 marks]**

```
// Apply only top(), push(), and pop() functions defined
// in C++ STL.

int performOperation(char opr, int op1, int op2)
{
   switch(opr) {
      case '+': return op1 + op2;
      case '-': return op1 - op2;
      default : return 0; }
}

int evaluatePostfix( string postfix )
{
    stack <char> aStack;
    int i, op1, op2, result = 0;

    while (i < postfix.length()) {
        char ch = postfix[i];
        if (isdigit(ch))
            // TODO : Question 1(c)(i)
            // Push ch to aStack
            // ...
        else if (isOperator(ch)) {
            // TODO : Question 1(c)(ii)
            // Assign the topmost two operands of aStack
            // to op1 and op2
            // ...

            // TODO : Question 1(c)(iii)
            // Trigger a call to performOperation()
            // ...

            // TODO : Question 1(c)(iv)
            // Push result to aStack
            // ...
        }
        i++;
    }
    // TODO : Question 1(c)(v)
    // Return result to function call
    // ...
}
```

Continued .....

Question 2

(a) The following recursive `binarySearch()` function searches for a `target` from `anArray[first]` through `anArray[last]`.

Complete the missing parts in your answer booklet. **[3 marks]**

```
// A binary search returns either an index such that
// anArray[index] == target, or -1 if the target is not
// an element of anArray[]
int binarySearch(const int anArray[], int first,
                                    int last, int target)
{
    int index;
    if (first > last)
        index = -1;
    else
    {
        int mid = first + (last - first) / 2;
        if (target == anArray[mid])
            // TODO : Question 2(a)(i)
            // ...
        else if (target < anArray[mid])
            // TODO : Question 2(a)(ii)
            // ...
        else if (target > anArray[mid])
            // TODO : Question 2(a)(iii)
            // ...
    }
    return index;
}
```

(b) Convert the following infix expression to its postfix equivalence.

```
( ( a - b ) - c * ( d / e ) ) + f
```

Trace and show the status of the stack after each step in your answer booklet based on the following table. **[4 marks]**

Character (input)	**Stack Content** (top to bottom)	**Postfix Expression** (output)
(	(	
:	:	:

Continued .....

(c) Provided with the following C++ program implemented using the `Queue` functions in Standard Template Library

(i) Describe the task performed by each `operateQueue1()` and `operateQueue2()`.

(ii) Justify if `operateQueue2()` is a valid recursive function.

(iii) Identify the program output for the given `main()` function.

[3 marks]

```
...

void operateQueue1(queue<int> q)
{
    while (!q.empty()) {
        cout << q.front() << " ";
        q.pop();
    }
}

void operateQueue2(queue<int>& q)
{
    if (q.empty())
        return;

    int data = q.front();
    q.pop();
    operateQueue2(q);
    q.push(data);
}

int main()
{
    queue<int> myQueue;
    myQueue.push(20);
    myQueue.push(25);
    myQueue.pop();
    myQueue.push(myQueue.back());
    myQueue.push(myQueue.front());
    myQueue.push(myQueue.front() + myQueue.back());
    operateQueue1(myQueue);
    operateQueue2(myQueue);
    operateQueue1(myQueue);

    return 0;
}
```

Continued .....

Question 3

(a) Sorting is a process of organizing a collection of data into either ascending, or descending order.

(i) Name the sorting algorithm exercised by the `sortAlgorithm()` function, and indicate also the sorting order. **[1 mark]**

```
// swap() is the built-in function in the C++ STL that
// that swaps the values of two variables

void sortAlgorithm(int anArray[], int n)
{
    int i, j;
    for (i = 0; i < n - 1; i++)
        for (j = 0; j < n - i - 1; j++)
            if ( anArray[j + 1] < anArray[j] )
                swap( anArray[j + 1], anArray[j] );
}

int main()
{
    int anArray[] = {64, 4, -11, 900};
    int n = sizeof(anArray) / sizeof(anArray[0]);

    sortAlgorithm(anArray, n);

    ...
}
```

(ii) Trace the `sortAlgorithm()` function while sorting the unsorted array `int anArray[] = {64, 4, -11, 900}`. Show all steps. **[3 marks]**

(iii) Re-implement the `sortAlgorithm()` function using a recursive formulation. **[2 marks]**

Continued .....

(b) Consider the following two hash functions:

`h1(key) = key % 13`, and `h2(key) = 7 - (key % 7)`.

To resolve collisions, the double hashing is performed as follows:

```
( h1(key) + i * h2(key) ) % TABLE_SIZE
```

where `TABLE_SIZE` is the pre-fixed hash table size, and `i` is increased from `i = 1, 2, 3, ...` , when collision occurs.

For `TABLE_SIZE = 13`, determine the hash table indexes for keys `14`, `75`, `53` and `10`, following the given sequence. **[4 marks]**

Continued .....

Question 4

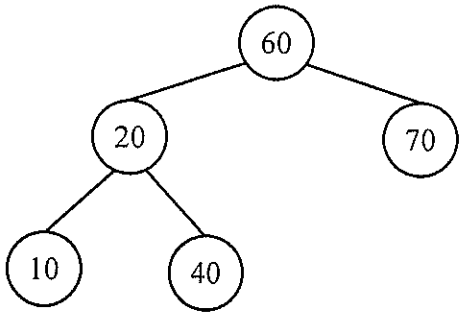
(a) The following `insertNode()` function inserts new leave nodes, one by one, to construct a binary search tree.

Complete the missing parts in your answer booklet. **[3 marks]**

```
...

struct node {
    int item;
    struct node *left, *right;
};

// Create a new binary search tree (BST) node
node* newNode(int item) {
    node *temp = new node;
    // TODO : Question 4(a)(i)
    // ...
    return temp;
}

// Insert a new node with newItem to the existing BST
node* insertNode(node* node, int item) {
    // If BST is empty, return a new node
    if (node == NULL)
        return newNode(item);
    // Otherwise, recur down
    if (newItem < node -> item)
        // TODO : Question 4(a)(ii)
        // ...
    else if (newItem > node -> item)
        // TODO : Question 4(a)(iii)
        // ...
    return node;
}

int main() {
    node *root = NULL;
    root = insertNode(root, 60);
    insertNode(root, 20);
    insertNode(root, 70);
    :
    return 0;
}
```

Continued .....

(b) Consider the following binary search tree.



(i) Show the tree resulted after inserting the new nodes 80, 75, 45, 65, 35, and 25, in that order. **[3 marks]**

(ii) Based on your answer in Question 4(b)(i), show the tree after removing node 40, followed by node 60. **[2 marks]**

(c) The following code snippet is the extension to the one in Question 4(a).

Name the traversal algorithm exercised by the `traversalAlgorithm()` function, and list the sequence of nodes visited based on your answer in Question(b)(ii). **[2 marks]**

```
// Visit every BST node following either preorder,
// inoder, or post-order traversal
void traversalAlgorithm(node *root) {
    if (root != NULL) {
        traversalAlgorithm(root->left);
        cout << root->item << "  ";
        traversalAlgorithm(root->right);
    }
}

int main()
{
    node *root = NULL;
    root = insertNode(root, 60);
    insertNode(root, 20);
    insertNode(root, 70);
    :
    traversalAlgorithm(root);
    return 0;
}
```

End of Page